The American University in Cairo

Spike Sorting Project

CSCE 3611 – Digital Signal Processing

Seif Eldawlatly

Farah Kabesh – 900191706

Karim Sherif - 900191474

Sunday 10th December 2023

Project Summary:

The aim of this project is to implement the Spike Sorting algorithm on a data text file that contains multiple columns, where each column represents the raw extracellular activity of an electrode. Spike Sorting is necessary to identify the activity of different neurons recorded on the same electrode. The sampling rate of the data is given as 24414 Hz. The code was written using Python programming language. This project required the following:

- Detecting spikes (by computing a threshold equal to 3.5 times the standard deviation of the first 500 samples of each electrode).
- Aligning spikes based on their peak value as well as ensuring that the duration of extracted spikes is 2 msec, where the peak is at the center of the extracted spike window.
- Extracting two features which are: the standard deviation of the samples of the spike, and the maximum difference between two successive samples of the spike.
- Showing the feature space obtained for each electrode after extracting the two features.
- Identifying the number of clusters by visually inspecting the feature space of each electrode.
- A figure to show the first 20,000 samples with an "*" to mark the detected spikes in different colors depending on the neuron each spike belongs to.
- A figure showing the average spike of each neuron.

Description of the Approach Used:

The "Data.txt" file is read and loaded using the *np.loadtxt()* function, where the values of each column are assigned to electrode 1 and electrode 2 respectively.

The threshold values for each electrode are then obtained by computing the standard deviation of the first 500 samples multiplied by 3.5. The standard deviation is calculated using *np.std()*.

In order to detect spikes and extract the peaks, the code iterates over the data of each electrode starting from the 500th sample and an if-statement is used to check if each current value of the electrode is greater than the threshold. If this condition is satisfied, the value is added to the *Window* list using *np.append()*. An else-if statement is then used to check if the *Window* list is not empty. This is because if the current value is less than the threshold and the *Window* list is not empty, this indicates that the spike has ended and the maximum value in *Window* represents the peak spike and can be obtained using *np.max()*. The peak values and their corresponding indices are respectively added and stored to the *peakSpike* and *markerIndex* lists.

The feature space, which is the which is the number of samples before and after each peak, is then computed by calculating the given sampling rate (24414 Hz) multiplied by the given duration (2 msec) and then dividing by 2.. After this, we iterate over the electrode to check whether the current value is equal to one of the peaks stored in the *peakSpike* list earlier. If this condition is satisfied and a match is found, a slice of the data is extracted where the peak is at the center, and the length of the data before and after the peak is equal to the number of samples obtained for the feature space. These extracted data slices are stored in the *samples* list.

The features that need to be extracted for each spike are: the standard deviation of the sample and the maximum difference between two consecutive values in the sample. These are computed with the aid of the *np.std()* and *np.max()* functions. The values for the standard deviation and maximum difference are added and stored as a pair *[sd, md]* in the *peakFeature* list.

A scatter plot of the features extracted is created with the standard deviation on the x-axis and the maximum difference on the y-axis. The elbow method is applied in order to determine the number

of clusters to be used for K-means clustering which is used to identify the spikes belonging to different neurons. The elbow method involves the use of *kmeans.inertia_* from the *sklearn.cluster* Python library and it is the Sum of Squares Errors (SSE) that calculates the sum of distances of all points in a cluster from the central point (squaring the sum of values minus the means). By visually observing the graph, the elbow point is identified which indicates the number of clusters that will be used to perform K-means clustering.

The *KMeans()* function takes the number of clusters determined earlier, and a scatter plot of the clustered data is created to visualize the different clusters of the spikes based on their features. Each point is represented by a color corresponding to its cluster.

A line plot is created to plot the first 20,000 samples of each electrode. Scatter points are then plotted with an "*" label to mark the detected spikes in different colors depending on the neuron each spike belongs to.

Finally, the average spikes of each neuron are plotted by iterating over the *markerIndex* list created earlier and separating the spike peaks based on their cluster label.

Outputs of the Project as Described in the Deliverables:

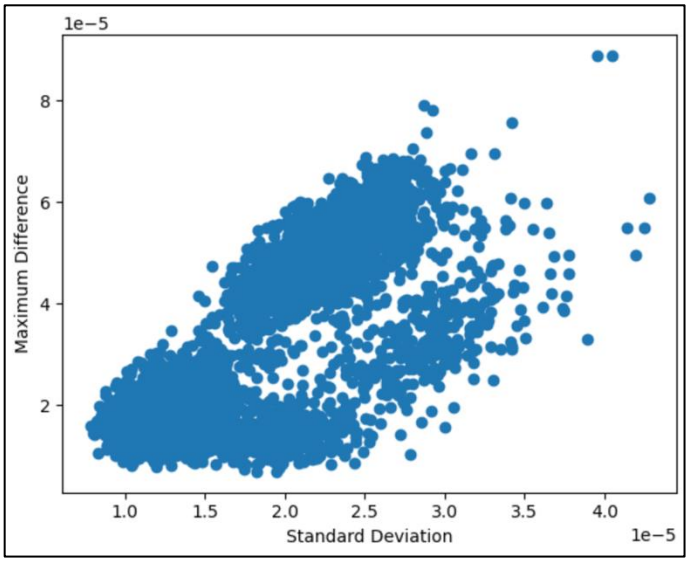
Electrode 1

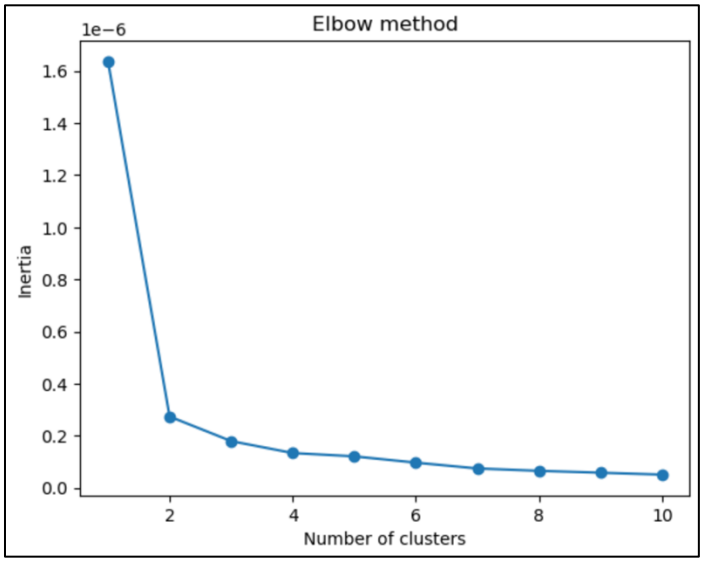
Feature Space:

24 samples before and after peak.

Number of Clusters Identified:

2 clusters.





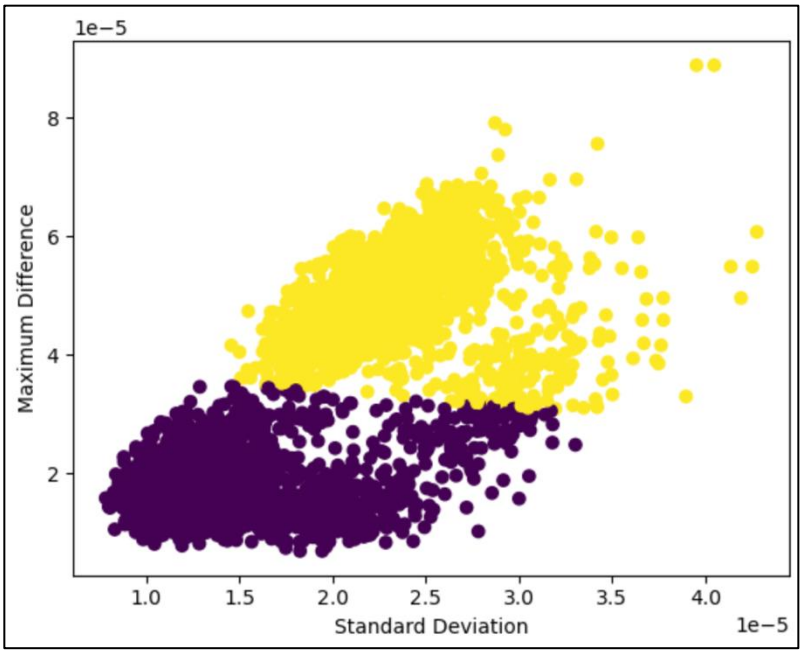


Figure showing the first 20,000 samples of the raw data of each channel with an "*" marking the detected spikes in different colors depending on neuron it belongs to:

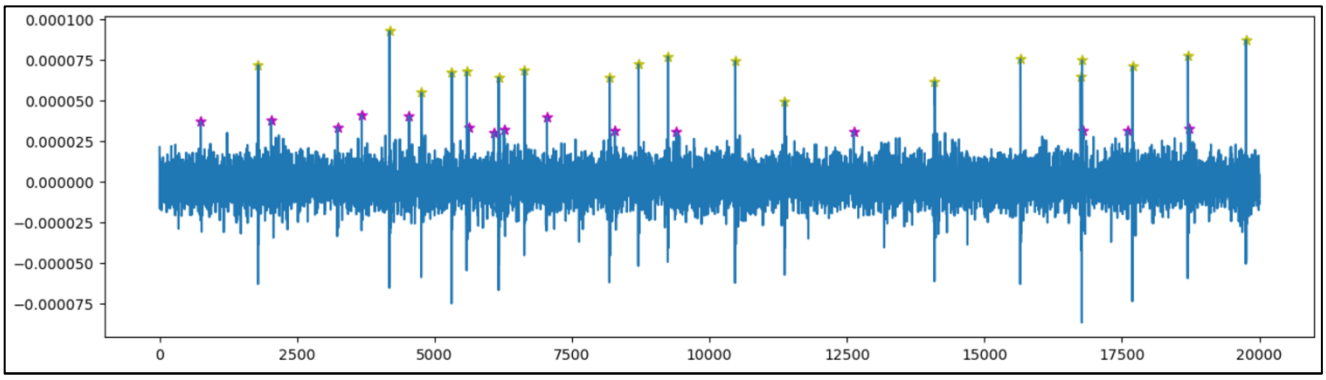
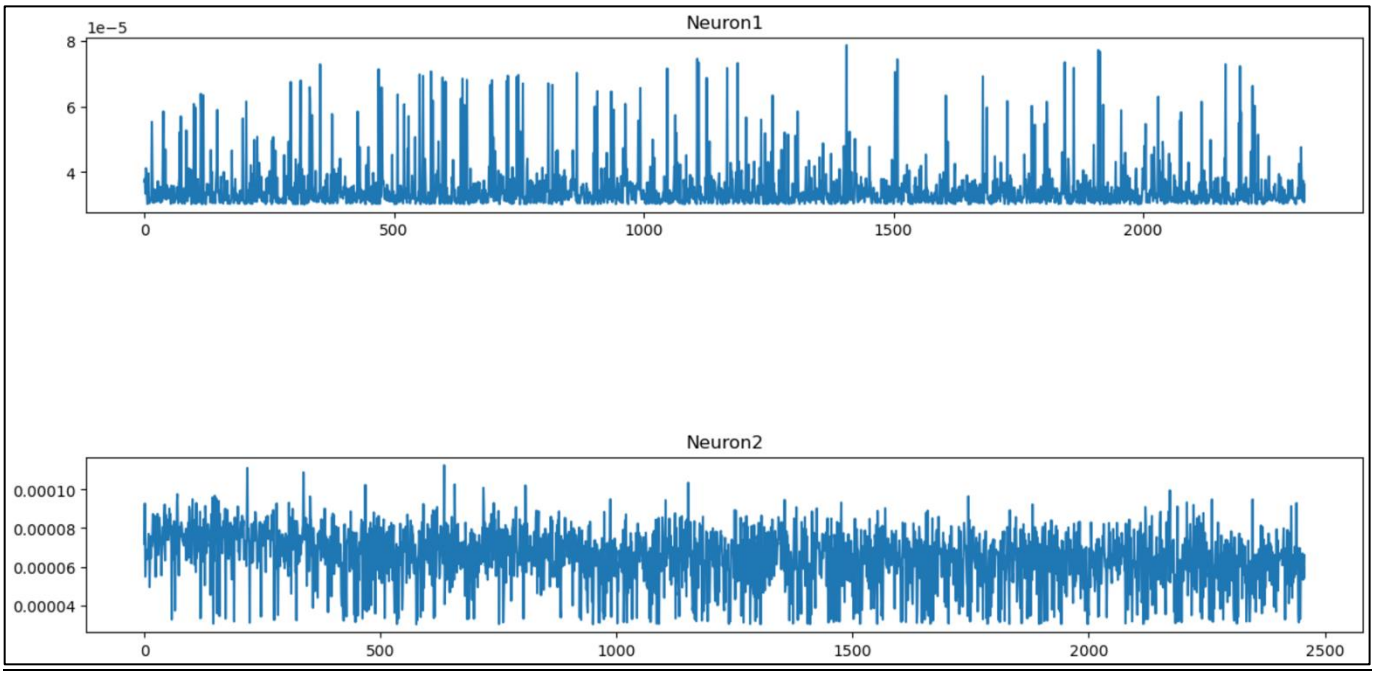


Figure showing the average spike of each neuron:



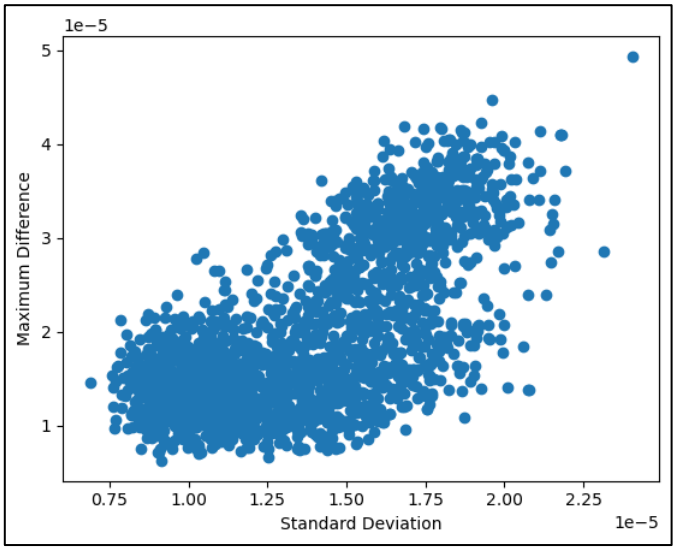
Electrode 2

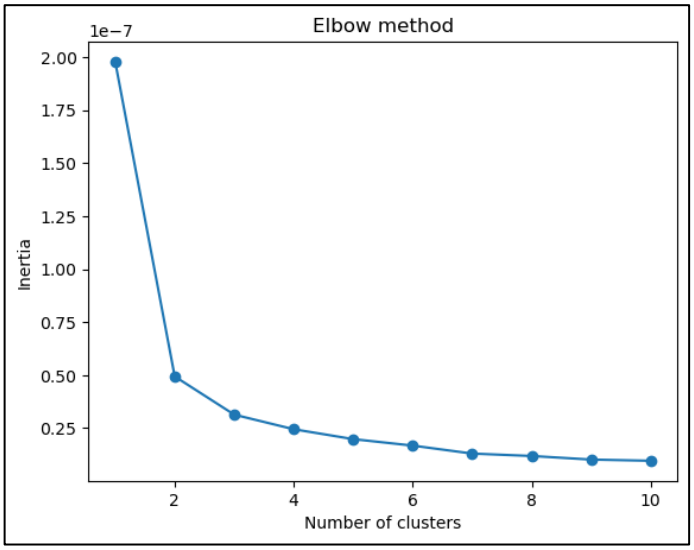
Feature Space:

24 samples before and after peak.

Number of Clusters Identified:

2 clusters.





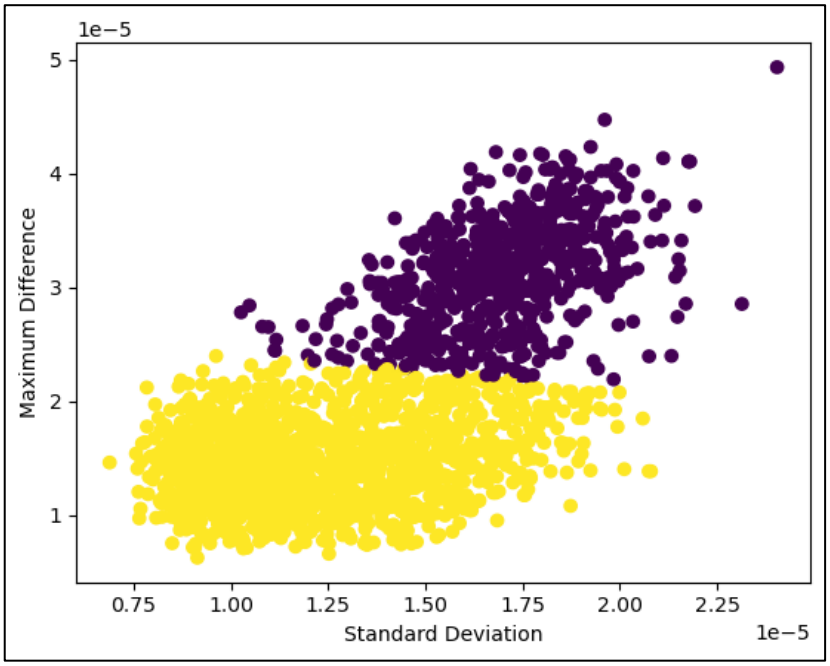


Figure showing the first 20,000 samples of the raw data of each channel with an "*" marking the detected spikes in different colors depending on neuron it belongs to:

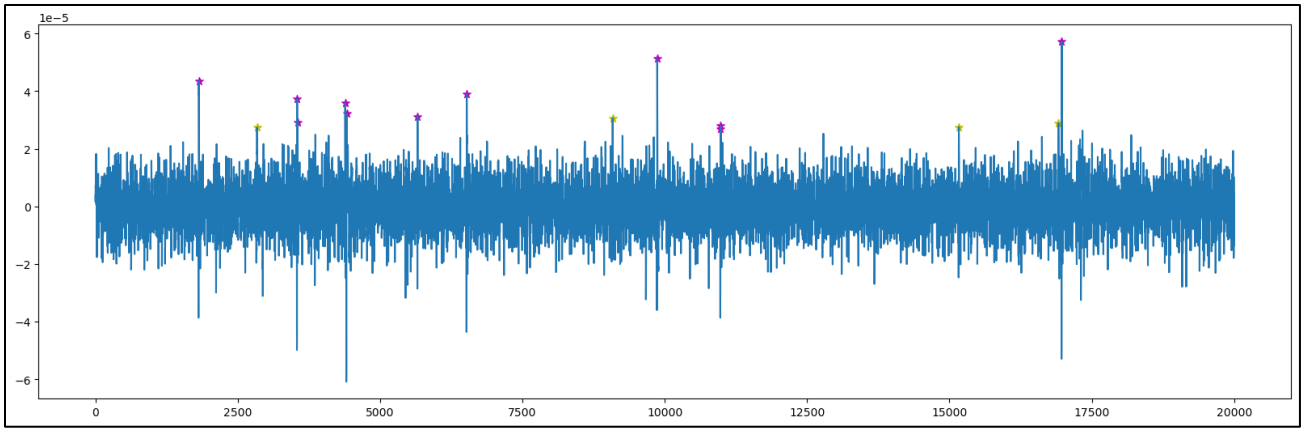


Figure showing the average spike of each neuron: